Handling Conflicts through Multi-level Editing in Peer-to-peer Environments

Claudia Lavinia Ignat, Moira Norrie, Gérald Oster

Handling Conflicts through Multi-level Editing in Peer-to-peer Environments

Claudia-Lavinia Ignat, Moira C. Norrie
Institute for Information Systems,
ETH Zurich
CH-8092 Zurich, Switzerland
{ignat,norrie}@inf.ethz.ch

Gérald Oster
Université Henri Poincaré Nancy 1, LORIA
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy Cedex, France
oster@loria.fr

ABSTRACT
The handling of conflicts in collaborative editing over hierarchical documents in peer-to-peer networks is a very important issue. In this paper we show how the multi-level editing approach that recursively applies a tombstone transformational approach over the document levels offers support for flexible conflict definition and resolution.

Author Keywords
multi-level editing, operational transformation, tombstone transformation functions, conflict handling

ACM Classification Keywords
I.7.1 [Document and Text Processing]: Document and Text Editing; H.5.3 [Information Interfaces and Presentation]: Group and Organisation Interfaces - Computer-supported cooperative work; H.1.2 [Models and Principles]: User/machine Systems - Human Factors

INTRODUCTION
Collaborative editing systems support a group of people editing a set of documents over a computer network. Documents subject to collaboration can be of various types, such as text, XML and graphical. A hierarchical document model encompasses a wide range of documents and offers support for semantically structured documents and in this paper we analyse collaboration over hierarchical structures of documents.

As we want to allow users to work disconnected from the network we have used, a replicated architecture where users edit copies of the document. Operational transformation [2] is a suitable approach for maintaining consistency between copies of the shared document. In [1, 6, 3] various operational transformation approaches for maintaining consistency over tree structures of the document have been proposed.

Members of a team should have the possibility to synchronise their work between pairs of individuals and, therefore, we wanted to develop a general synchronisation mechanism that works in peer-to-peer networks. Some of the existing operation transformational approaches such as FORCE [8] and SAMS [6] rely on a central repository where changes done by users are published and a user can synchronise their local changes only against the repository. These approaches are limited to asynchronous collaboration over a shared repository and they cannot be applied for peer-to-peer communication.

Most of the approaches for maintaining consistency adopt an automatic way for merging concurrent changes and they do not deal with conflicts. In FORCE [8] a flexible merging approach has been proposed where rules for the detection of conflicts between pairs of operations are specified. However, this approach adopts automatic merging and local changes are adopted if conflict occurs. Users are not allowed to manually choose between changes that are in conflict. Moreover, the FORCE approach is not applicable in peer-to-peer networks.

Approaches for merging hierarchical documents such as SGML [1], XML and CRC (Class, Responsibility, Collaboration) documents [6] and the treeOPT approach described in [3] for merging any hierarchical documents also adopt an automatic solution for merging and do not allow the manual intervention of users. Moreover, some specific rules for the concurrent deletion of a node in the tree and modifications referring to that node are considered. The merging approaches described in [6] and [3] consider that, once a node in the tree is deleted, any concurrent changes referring to that node are discarded. The merging approach in [1] excises the deleted nodes, but keeps them as separate structures. In this way, concurrent operations to the deletion of the node and targeting the deleted node are executed. However, the approach does not explain when and how the excised nodes are reassembled into the initial tree document. We therefore see the need of an approach that keeps the deleted nodes and considers changes performed on them during the merging process. A resolution mechanism would present conflicts to the users and their decision would determine whether or not the delete nodes are reassembled in the initial document.

Dealing with conflicts in an easy manner requires a suitable model of the document. Even if the structure of the document is hierarchical, the operational transformation adopted by most existing approaches [6, 1] is similar to the approach for linear structures and does not take advantage of the tree structure of the document. The existing operation-based approaches maintain a single history buffer where the executed operations are kept. Operations are not associated with the structure of the document and therefore it is difficult to select which operations refer to which node in the document. This fact has limitations for the definition and resolution of

conflicts. For instance, they do not allow the possibility of defining that any operations that refer to the same node are conflicting and the user can later choose one of the versions of the node. To determine which operations from the history buffer refer to which node is very complex, since the structure of the document is dynamically changed with the execution of each operation.

Multi-level editing used in the treeOPT approach [3] involves logging edit operations that refer to each node. In this way, conflicting operations that refer to the same subtree of the document are easily detected by the analysis of the histories associated with the nodes belonging to the subtree. Therefore, the resolution of conflicts is simplified in comparison to the approaches using a single history buffer.

In this paper we present an extension of the treeOPT approach that handles conflicts in an appropriate way including the ones involving concurrent deletions of nodes and modifications of those nodes. The treeOPT approach recursively applies an operational transformation algorithm working for linear structures over the hierarchical document levels. Our method was to combine treeOPT with a linear operational transformation algorithm that works in peer-to-peer networks and maintains the structure of the deleted nodes. As SOCT2 together with the tombstone transformation functions [7] is an operational transformation algorithm that works in peer-to-peer networks and the tombstone transformational approach deals with tombstone entities, we decided to combine treeOPT with SOCT2 and the tombstone transformational approach. In this paper we show how the multi-level editing approach together with the tombstone transformational approach supports dealing with conflicts.

We start by presenting in the next section the treeOPT approach. We then go on by describing in the third section the operational transformation approach that deals with tombstone entities. The fourth section of this paper presents how the adaptation of the treeOPT approach with the tombstone transformational approach offers users a flexible way of handling conflicts. In the last section we provide some concluding remarks and some future work directions.

THE TREEOPT APPROACH

The treeOPT approach can be applied for maintaining consistency for collaboration over hierarchical structured documents, such as textual [3] and XML [5] documents. XML documents conform to a tree model by definition. Text documents can also be structured using a hierarchical model. A text document is composed of a list of paragraphs, each paragraph as a list of sentences, each sentence as a list of words and each word as a list of characters. A document has associated various levels, such as the document (level 0), paragraph (1), sentence (2), word (3) and character (4) in the case of text documents. Each node in the hierarchical structure will keep a history of operations of insertions and deletions of child nodes. We present in what follows our definition for a node in the tree.

A node N of a document is a structure of the form $N =< parent, children, length, history, content >$, where

- $parent$ is the parent node for the current node
- $children$ is an ordered list $[child_1, ..., child_n]$ of child nodes
- $length$ is the length of the node in terms of the number of leaf nodes,

$$length = \begin{cases} 1, & \text{if } N \text{ is a leaf node} \\ \sum_{i=1}^{n} length(child_i), & \text{otherwise} \end{cases}$$

- $history$ is an ordered list of operations executed on child nodes
- $content$ is the textual content of the node

$$content = \begin{cases} aCharacter, & \text{if } N \text{ is a leaf node} \\ \sum_{i=1}^{n} content(child_i), & \text{otherwise} \end{cases}$$

We define the level of a node to be the height of the node, i.e. the length of the path from the root to the node.

An operation is specified by its *type*, *position* in the tree and its *content*. The type of an operation can be insertion or deletion. The position of an operation is represented by a vector of indexes specifying the path starting from the root to the node where the operation has been applied. The content of an operation is represented by the node that is inserted or deleted. An operation has associated a state vector [2] and the identifier of the site that generated the operation. The level of an operation is the level of the node in whose history the operation is kept.

After its execution, an operation is added to the history of the parent node of the node that is inserted or deleted. Due to the fact that the position of an operation is the path to the node where the operation is applied, we represented the position of an operation stored in a history buffer by the position of the target node relative to its parent. For instance, the operation with the absolute position representation *InsertWord(3,2,4,"collaborative")* of inserting the word *"collaborative"* in paragraph 3, sentence 2 as the 4th word, will be kept in the history associated with sentence 2 in paragraph 3 with the relative position representation *InsertWord(4,"collaborative")*. We are going to refer to an operation given by its absolute position as a composite operation and to an operation viewed at a certain level as a simple operation. Operations in history buffers are viewed as simple operations at the level of the node to which they belong. But, they can be viewed as composite operations taking into account the position of the node to which they belong. Composite operations are characterised by a position vector, while simple operations are characterised by a position index.

Local operations are immediately executed and added to the corresponding history buffers. For the integration of a remote operation into the corresponding history buffer, the treeOPT approach recursively applies an existing operational transformation algorithm, such as GOT [11], GOTO [10] or SOCT2 [9], over the hierarchical document structure. In what follows we present the implementation of the treeOPT algorithm using the SOCT2 algorithm.

Algorithm *treeOPT-SOCT2(O, RN, L)* {
  CN = RN;
  for (l := 1; $l \leq L$; l++) {
    O_{new} = $Composite2Simple$(*O*, *l*);
    EO_{new} = $SOCT2(O_{new}, history(CN))$;
    $position(O)[l]$ = $position(EO_{new})$;
    if ($level(O)$ = l) {
      Do O;
      $Append(EO_{new}, history(CN))$;
      Update $length(CN)$;
    }
    CN = $child_i(CN)$, where $i = position(EO_{new})$;
  }
}

Given a new causally ready composite operation, O, the root node of the hierarchical representation of the local copy of the document, *RN*, and the number of levels in the hierarchical structure of the document, L, the execution form of O is computed. In the case of text documents structured into paragraphs, sentences, words and characters, *L=4* and *RN=document*. Determining the execution form of a composite operation requires finding the elements of the position vector corresponding to a coarser or equal granularity level than that of the composite operation. For each level of granularity l, starting with paragraph level and ending with the level of the composite operation, the SOCT2 algorithm is applied to find the execution form of the corresponding regular operation. SOCT2 does not perform transformations on composite operations, but rather on simple ones. Therefore, we had to define the function *Composite2Simple*, that takes as arguments a composite operation, together with the granularity level at which we are currently transforming the operation, and returns the corresponding simple operation [4]. The operational transformation algorithm is applied to the history of the current node *CN* whose granularity level is *l-1*. Recall that, for example, to find the corresponding paragraph position, transformations need to be performed against the operations kept in the document history. The lth element in the position vector will be equal to the position of the execution form of the simple operation. If the current granularity level l is equal to the level of the composite operation, the composite operation is executed and the execution form of the simple operation is stored into the history buffer associated with the current node. Otherwise, the processing continues with the next finer granularity level, with *CN* being updated accordingly. The function *SOCT2(O, HB)* takes as parameters a causally-ready regular operation O and a history buffer *HB* and returns the execution form of O.

TOMBSTONE TRANSFORMATION FUNCTIONS

As previously mentioned, for dealing with conflicts in the face of concurrent delete operations that target nodes of the hierarchical structure, we adapt the treeOPT approach to mark deleted entities as invisible, but do not remove these entities from the structure of the document. To our knowledge, the only existing transformation functions that deal with tombstone entities are those proposed in [7]. These transformation functions are presented below:

T($ins(p_1, el_1, sid_1), ins(p_2, el_2, sid_2)$){
  if ($p_1 < p_2$) return $ins(p_1, el_1, sid_1)$
  else if ($p_1 = p_2$ and $sid_1 < sid_2$) return $ins(p_1, el_1, sid_1)$
  else return $ins(p_1 + 1, el_1, sid_1)$
}

T ($ins(p_1, el_1, sid_1), del(p_2, sid_2)$) {
  return $ins(p_1, el_1, sid_1)$
}

T ($del(p_1, sid_1), ins(p_2, el_2, sid_2)$) {
  if ($p_1 < p_2$) return $del(p_1, sid_1)$
  return $del(p_1 + 1, sid_1)$
}

T ($del(p_1, sid_1), del(p_2, sid_2)$) {
  return $del(p_1, sid_1)$
}

Besides their adaptation for maintaining the structure of deleted nodes, these transformation functions have some other advantages. First, the definition of inverses of the forward transformation functions [7] required by the SOCT2 algorithm is straightforward since they are injective functions. Second, as far as we are aware, these transformation functions are the first ones that satisfy both mandatory consistency properties [9] named C_1 and C_2. Consequently, our treeOPT-SOCT2 algorithm is the first operational transformation algorithm applicable in peer-to-peer environments that maintains consistency of replicated hierarchical documents.

DEALING WITH CONFLICTS

We define two concurrent operations as being in conflict if they modify the same conflict node or any of its children. Conflict nodes can be defined by specifying a certain level of granularity for conflict in which case all nodes of that granularity level are considered conflict nodes. Certain nodes in the hierarchical structure can also be specified as conflict nodes.

When an operation is integrated into the document structure, the tree is traversed from top to bottom till the level of the operation is reached and the operation can be integrated into its corresponding history buffer. If during this integration a conflict node is encountered, a check is done to see if concurrent operations have been performed on the conflict node. Detection of conflict is done by the analysis of the histories associated with conflict node and all its child nodes. If the history buffers distributed throughout the subtree rooted at the conflict node contain an operation concurrent with the remote operation, a conflict is detected and according to a resolution policy the conflict is resolved. The resolution policy can be automatic or manual. In the automatic resolution policy, the operations that win a conflict are decided according to certain rules. For instance, the operations that win a conflict can be decided according to priorities assigned to the sites that generated the operations. The manual resolution policies require user intervention for choosing between conflicting operations.

If an integration algorithm is chosen that deletes the nodes in the hierarchy once a deletion was performed, conflicts between the deletion and any other operation concurrently performed on the deleted subtree cannot be detected. We therefore adopted a transformation algorithm that does not delete nodes, but marks them for deletion. In what follows we analyse the cases that show the benefit of choosing to mark nodes for deletion instead of their physical deletion. These cases are the ones involving the deletion of a node

of the tree and the concurrent insertion inside the node that is deleted. These two cases are distinguished by the order of execution of the two concurrent operations, i.e. insertion followed by deletion or deletion followed by insertion. Suppose that the level of the insertion operation is l_1 and that of the deletion operation is l_2. The cases that need special attention are the ones where $l_1 > l_2$.

We analyse first the case when the deletion operation was executed and the concurrent insertion operation has to be integrated. When deletion is performed, the node targeted by deletion is marked as invisible, but further considered when a remote operation has to be integrated. When a remote insertion operation has to be integrated, ancestors of nodes to be deleted and marked as conflict nodes will be detected during processing, showing that the delete and remote insert are concurrent. If a conflict node is a descendant of the node to be deleted and an ancestor of the node to be inserted, conflict is also detected when the processing reaches the conflict node. Processing is also called for the deleted nodes and when processing reaches the conflict node, conflict is detected as the conflict node is marked as deleted.

The second case to be analysed is the one when the insertion operation is executed first and the concurrent deletion operation has to be integrated. If the conflict node is an ancestor of the node to be deleted, conflict is detected when the conflict node is reached in the processing phase. The insert operation in one of the histories distributed throughout the tree rooted at the conflict node will be detected as being in conflict with the remote delete operation. After a deletion is performed, checking for conflicts has to be performed also on the deleted subtree. Processing is therefore further performed and if a conflict node is detected during processing and the conflict node is an ancestor of the inserted node, the conflict is detected as the insertion operation is performed on a node marked as deleted.

If deleted nodes are removed from the document, concurrent changes performed on the deleted parts of the tree are not considered and therefore no conflicts are presented to the user. By using the tombstone transformational approach, the operations concurrent with deletions of nodes in the document and targeting concurrently deleted parts of the document are performed. In this way, if a conflict node is set that detects the conflict between deletion and the concurrent change, users are presented with the two conflicting versions of the conflict node: first one showing the deleted parts of the document (for e.g. by lines through deleted text) and the second one showing concurrent modifications done on the conflict node.

For the resolution method, an undo mechanism has to be provided to cancel some operations performed. An undo mechanism for the SOCT2 algorithm using the TTF transformation functions has been provided in [12] and we are currently working on the integration of the undo mechanism with treeOPT.

CONCLUSIONS AND FUTURE WORK

In this paper we presented how the application of the tombstone transformational approach recursively over the levels of the document by using the treeOPT approach offers support for conflict handling in peer-to-peer networks. The paper also shows that the tombstone transformational approach is general and can be applied for hierarchical structures. We are currently investigating the multi-level editing undo and building a prototype based on the ideas presented in this paper.

REFERENCES

1. Davis, A. H., Sun, C., and Lu, J. Generalizing Operational Transformation to the Standard General Markup Language. *Proc. of CSCW'02*, New Orleans, Louisiana, USA, Nov. 2002, 58–67.
2. Ellis, C. A., and Gibbs, S. J. Concurrency Control in Groupware Systems. *Proc. of ACM SIGMOD'89*, Seattle, Washington, USA, 1989, pp.399-407.
3. Ignat, C.-L., and Norrie, M.C. Customizable Collaborative Editor Relying on treeOPT Algorithm. *Proc. of ECSCW'03*, Helsinki, Finland, Sept. 2003, 315–334.
4. Ignat, C.-L., and Norrie, M.C. Tree-based Model Algorithm for Maintaining Consistency in Real-time Collaborative Editing Systems, *Workshop on Collaborative Editing*, CSCW'02, New Orleans, Louisiana, USA, Nov. 2002.
5. Ignat, C.-L., and Norrie, M.C. Flexible Collaboration over XML Documents. *Proc. of CDVE'06*, LNCS vol. 4101, Mallorca, Spain, Sept. 2006.
6. Molli, P., Skaf-Molli, H., Oster, G., and Jourdain, S. SAMS: Synchronous, Asynchronous, Multi-Synchronous Environments. *Proc. of CSCWD'02*, Rio de Janeiro, Brazil, Sept. 2002, 80–85.
7. Oster, G., Urso, P., Molli, P., and Imine, A. Proving Correctness of Transformation Functions in Collaborative Editing Systems. *Research Report* RR-5795, INRIA Lorraine, LORIA, Dec. 2005.
8. Shen, H., and Sun, C. Flexible Merging for Asynchronous Collaborative Systems. *Proc. of CoopIS'02*, Irvine, California, USA, Nov. 2002, 304–321.
9. Suleiman, M., Cart, M., and Ferrié, J. Serialization of Concurrent Operations in a Distributed Collaborative Environment. *Proc. of GROUP'97*, Phoenix, Arizona, USA, Nov. 1997, 435–445.
10. Sun, C., Ellis, C. Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements. *Proc. of CSCW'98*, Seattle, Washington, USA, Nov. 1998, 59–68.
11. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D. Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems. *ACM. TOCHI*, vol. 5, issue 1, 1998, 63–108.
12. Weiss, S. Annulation de groupe dans les éditeurs collaboratifs. *Master Thesis*, Université Nancy 1, June 2006.